2015-06

Inferring the presence of reverse proxies through timing analysis

Alexander, Daniel R.

Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/45803

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

INFERRING THE PRESENCE OF REVERSE PROXIES THROUGH TIMING ANALYSIS

by

Daniel R. Alexander

June 2015

Thesis Co-Advisors:	Geoffrey Xie
	Robert Beverly

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704–0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE 06-19-2015	3. REPORT TYPE AND DATES COVERED Master's Thesis 07-05-2013 to 06-19-2015	
4. TITLE AND SUBTITLE INFERRING THE PRESENCE OF REVERSE PROXIES THROUGH TIMING ANALYSIS			5. FUNDING NUMBERS 1127506
6. AUTHOR(S) Daniel R. Alexander			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Science Foundation 4201 Wilson Blvd., Arlington, VA 22230			10. SPONSORING / MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis presents a method for inferring the presence of a reverse proxy server using packet timing analysis from the vantage point of a client system. This method can determine whether Internet users are receiving web content from the actual source or from some potentially spoofed proxy device; leading to better risk assessment and understanding of the cyber terrain. By using only the measurement and comparison of three-way handshake and content request/delivery packet round trip times, we identify an accurate classifier that detects the presence of a reverse proxy server with over 98% accuracy. This is an improvement over other inference methods because all measurements can be done from an external client machine. A secondary yet significant contribution is the robust data set that was produced as a result of this research. We have collected a set of over 6 million data points from a known set of 30 globally dispersed machines, which was instrumental in our research efforts and will be used for further studies and exploration.

14. SUBJECT TERMS Active Measurement, Timing Analysis, Reverse Proxy, Machine Learning, Inference, Classification			15. NUMBER OF PAGES 65
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2–89) Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

INFERRING THE PRESENCE OF REVERSE PROXIES THROUGH TIMING ANALYSIS

Daniel R. Alexander
Major, United States Army
B.A., Colorado State University, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2015

Author:	Daniel R. Alexander
Approved by:	Geoffrey Xie Thesis Co-Advisor
	Robert Beverly Thesis Co-Advisor
	Peter J. Denning Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis presents a method for inferring the presence of a reverse proxy server using packet timing analysis from the vantage point of a client system. This method can determine whether Internet users are receiving web content from the actual source or from some potentially spoofed proxy device; leading to better risk assessment and understanding of the cyber terrain. By using only the measurement and comparison of three-way handshake and content request/delivery packet round trip times, we identify an accurate classifier that detects the presence of a reverse proxy server with over 98% accuracy. This is an improvement over other inference methods because all measurements can be done from an external client machine. A secondary yet significant contribution is the robust data set that was produced as a result of this research. We have collected a set of over 6 million data points from a known set of 30 globally dispersed machines, which was instrumental in our research efforts and will be used for further studies and exploration.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 INTRODUCTION **1**
1.1 Problem Statement. 2
1.2 Research Description and Hypothesis 4
1.3 Organization 5

2 BACKGROUND AND RELATED WORK **7**
2.1 Fundamental Knowledge 7
2.2 Previous Work 10

3 METHODOLOGY **15**
3.1 Data Collection 15
3.2 Data Analysis. 22

4 RESULTS AND ANALYSIS **25**
4.1 Data Set Acquired 25
4.2 Initial Training Set Results 29
4.3 Performance of Initial Classifier on Test Set. 30
4.4 Adjusting the Methodology 33
4.5 Evaluating the Strength of the Classifier 34

5 CONCLUSIONS AND FUTURE WORK **39**
5.1 Conclusions 39
5.2 Future Work 40

List of References **43**

Initial Distribution List **47**

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

Figure 2.1 Transmission Control Protocol (TCP) Three-Way Handshake (3WHS) timing diagram . . . 8

Figure 2.2 Hypertext Transfer Protocol (HTTP) timing diagram . . . 9

Figure 3.1 Map of PlanetLab nodes used . . . 16

Figure 3.2 The three different instances of timing measurement configurations 17

Figure 3.3 Permutation of a web request iteration . . . 20

Figure 3.4 Side by side comparison of web request by direct connection and via reverse proxy . . . 22

Figure 4.1 Comparison of GET/3WHS ratios . . . 28

Figure 4.2 Initial division of the data into training and testing sets . . . 29

Figure 4.3 Modified subset method on data set . . . 34

Figure 4.4 Graph of classifier performance when adjusted by 10% higher and lower . . . 35

Figure 4.5 Use of Cumulative Distribution Function (CDF) to adjust acceptable threshold to meet objective . . . 36

Figure 4.6 Graph of adjusted classifier performance on reverse proxies remotely connected to the web servers . . . 38

Figure 4.7 Graph of adjusted classifier performance on reverse proxies locally connected to the web servers . . . 38

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

Table 3.1 Confusion matrix template 23

Table 4.1 Summary of flows captured 26

Table 4.2 Sample of data in table of captured flows 29

Table 4.3 Confusion matrix of classifier on test set 30

Table 4.4 Comparison of evaluation of classifier on initial 80/20 split and after 5-fold cross validation with classification tree learning 32

Table 4.5 Final results of classifier performances with sample bias removed 34

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

3WHS	Three-Way Handshake
CDF	Cumulative Distribution Function
CDN	Content Distribution Network
CPU	Central Processing Unit
DMZ	Demilitarized Zone
DNS	Domain Name Server or Domain Name Service
DoD	Department of Defense
DoS	Denial of Service
DDoS	Distributed Denial of Service
FN	False Negative
FP	False Positive
HIPAA	Health Insurance Portability and Accountability Act of 1996
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
IPv6	Internet Protocol version 6
ISP	Internet Service Provider
MTU	Maximum Transmission Unit

NDSS	Network and Distributed System Security
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
NPS	Naval Postgraduate School
NPV	Negative Predictive Value
PDU	Protocol Data Unit
PPV	Positive Predictive Value
RFC	Request For Comments
RTT	Round Trip Time
US	United States
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
TSO	TCP Segmentation Offload
UDP	User Datagram Protocol
XML	Extensible Markup Language

Acknowledgments

I wish to express my sincere appreciation to Dr. Geoffrey Xie and Dr. Robert Beverly, whose patient guidance and advice have been invaluable. They have both, each in their own way, continued to raise the bar and inspire me to reach it every time. I consider myself very lucky to have had the opportunity to learn from such brilliant professionals.

I'd like to thank the National Science Foundation for their role in supporting this thesis. Their efforts to advance research and education in science and engineering will surely continue to make America great.

I'd also like to thank, and congratulate my peers in Cohort 368-141. Many hours of pain and suffering were shared as well as many more late-night and weekend communications. Their camaraderie, collaboration, humor, and friendship made this experience thoroughly rewarding. It is unfortunate that none of them are members of the best branch of the military but if they decide to visit, I will have a reflective belt ready for them.

Finally, and most importantly, I'd like to acknowledge with gratitude, the support and love of my wife and children. Their encouragement throughout this process was pivotal to the success of this thesis and to every other challenge put before me. The sacrifices they make are greatly appreciated and to them I owe a most profound debt of gratitude.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: INTRODUCTION

Any attempt at ensuring that a given interaction between two parties is legitimate must be built upon the confidence that those involved are either the expected participants or their designated representatives. This is evident in just about every aspect of our lives. Banks require proof of identity before accepting or paying out money to customers. Libraries require identification before checking out books. Retail outlets are supposed to verify identity before accepting credit card payments. The point is that if identities cannot be verified then the information, products, or services received cannot be readily trusted to be authentic. Why then, are people so willing to accept data from web servers without a second thought as to whether they are getting it from an original, authentic source or from some intermediate proxy that could potentially be a malicious back end or spoofed device?

This is not to say that there are no protocols or methods that can protect users from blindly accepting web content. For example, digital certificates are used to provide mutual authentication between clients and servers on many of today's web sites. While these mechanisms exist, they are not ubiquitous. Many providers do not employ such technical solutions because of the added expense or complexity that comes with them.

The increasing global dependence on the Internet, as well as the expanding deployment of Internet Protocol version 6 (IPv6), is creating an environment where nearly every device imaginable can and does connect to the Internet. In fact, the number of IPv6 addresses available for devices is so astronomically huge that we will not likely ever see the day when every IPv6 address is allocated and in use. Nonetheless, the global community is connecting a colossal Internet of Things (IoT) at a blistering pace. We do not just connect our home computers anymore. We have managed to interconnect nearly every aspect of our lives, big and small. Devices such as home security systems, mobile devices, medical instruments, national defense systems, air traffic and energy control systems, all the way down to bathroom scales and coffee makers. Some estimates put the number of devices on the Internet between eight and ten billion today and predict that number to jump to forty billion devices by 2020 [1]. This increased connectivity between systems makes us all more dependent on them and their ability to provide data to anyone, anywhere, anytime. Along with that dependence comes an even greater need to defend those systems.

Just as crime rates are higher in cities with denser populations, attackers take advantage of the Internet's target rich environment. Mega data breaches, evolving targeted attacks, end-user attitudes toward social media, and vulnerabilities found in the plethora of Internet devices are only a few of the areas that are being exploited daily by malicious actors and governments. In 2013 alone, over 552 million identities were compromised "putting consumer's [sic] credit card information, birth dates, government ID numbers, home addresses, medical records, phone numbers, financial information, email addresses, login, passwords, and other personal information into the criminal underground" [2]. That is why it is becoming more and more imperative that not only us users, but our devices, know when we are actually communicating with who we think we are.

Much research has been devoted to source authentication in a variety of fields such as anthropology, law, retail, and computer science. While no single, foolproof solution has been found to provide absolute protection, we continue to innovate and discover methods to strengthen our defense-in-depth strategies.

This thesis builds upon one such concept that was introduced by a former student of the Naval Postgraduate School (NPS) [3]. It reinforces his findings and makes a significant contribution to Internet transaction risk assessment methods and topology discovery. This novel method can determine whether Internet users are receiving web content from the actual source or from some potentially spoofed proxy device; leading to better risk assessment and understanding of the cyber terrain.

1.1 Problem Statement

Users are constantly being served content by systems that they are unknowingly connected to despite employing sophisticated authentication measures and encryption schemes. One type of these systems, called reverse proxy servers, is used heavily throughout the Internet for both beneficial and malicious purposes.

By definition, a proxy is a person authorized to act for another [4] and in computer networks a proxy is a system that performs a service on behalf of another system. There are several types of proxies in use with the most common being a forwarding web proxy server. This type of server is usually installed at the edge of a computer network to retrieve web content from the Internet on behalf of the internal requesting clients. Conversely, reverse proxies provide data on behalf of the origin content provider. They do this by connecting to the content source providers and returning the data to the requesting client as if it had originated from the proxy itself. Reverse proxies are therefore used to add extra layers of security to publicly accessible services by providing content to requesting clients without exposing the origin servers.

While this provides an effective method of obscuring the origin servers, it is not their only purpose. They are also used for many other reasons beyond hiding the existence and characteristics of the origin servers such as improving performance and access control.

The ability of reverse proxies to reduce the work load and exposure of web servers is beneficial in many ways. Unfortunately, they also prevent users from being able to accurately calculate the inherent risk of transactions because they do not know where the original data is coming from. Reverse proxy servers can be installed at any point between a client and web server so it makes it difficult to confirm the device, geographical location, network, or organization that is actually providing the content.

Not knowing the origin of web content may not be a concern for most when it comes to casual web browsing however, it can be problematic in other situations. For example, a client can securely connect to a reverse proxy but the back-end connection to the web server may be unprotected. Whether it be due to negligence of the administrator or honest misconfiguration of the proxy, it can leave the entire session vulnerable.

The added obfuscation, improved performance, and access control that reverse proxies provide are all good features of an efficient and secure network. These devices are equally appealing to cyber criminals for many of the same reasons as they give those individuals the means to conduct their activities while hiding their true origins.

Malicious actors are very concerned with launching attacks without being detected. They do this often by using other compromised machines that will mislead and delay investigators, thus allowing for more time to accomplish their attacks. This is essentially what a reverse proxy does by design. It provides content to a requesting client while only allowing the client to see the traffic between itself and the reverse proxy. Anything that happens behind the proxy is unknown to the client.

The threat and scale of attacks are increased exponentially when many compromised machines are communicating through the Internet via some command channel. This defines a botnet and it is one of the primary platforms from which Internet attacks are launched today. Attacks such as Distributed Denial of Service (DDoS), spam, phishing, and identity theft are commonly launched from botnets and with an average annual cost of just over $3.5 million per U.S. company [5], they are very expensive. Botnets are often installed as hosting services and proxy unsuspecting clients to malicious content [6]. One example to consider is when a botnet has compromised a legitimate reverse proxy and changed the location from which it retrieves content. Users who visit the website employing the reverse proxy expect to be served their requested web page but instead are served illegal content or other malware.

1.2 Research Description and Hypothesis

Our hypothesis was that we could infer the presence of a reverse proxy server by analyzing the Transmission Control Protocol (TCP) flows from a requesting client's vantage point and comparing the Round Trip Time (RTT) of the Three-Way Handshake (3WHS) to that of the Hypertext Transfer Protocol (HTTP) content delivery. Because they have to forward requests to an internal device, reverse proxy servers showed an increased RTT for HTTP content in previous research [3]. The primary question this thesis sought to answer was, given a set of Internet Protocol (IP) addresses assigned to geographically distributed web servers known to be operating with and without reverse proxies, could a reliable threshold classifier learn to accurately detect those that are behind reverse proxy servers?

Subsidiary questions that we answered were:

1. What was the achievable performance (derived from the false positive and false negative rates) for such a program?
2. Was a packet RTT analysis alone sufficient to determine the presence of a reverse proxy server?
3. If so, what was the best classifier to determine the presence of a reverse proxy?

4. What is the impact of adjusting the optimal classifier by 10% higher or lower?

1.3 Organization

This thesis is organized into five chapters.

Chapter 1 is an introduction and describes the need that our thesis research aims to address. It explains the concept of the research and summarizes the methodology by which the primary question will be answered.

Chapter 2 describes the background and related work. We review previous research that deals with reverse proxy inference and explore other programs and methodologies similar to ours.

Chapter 3 pertains to our methodology and design considerations and how they apply to real world problems. We describe how we apply fundamental knowledge to the overall design to initiate content requests, handle TCP flows, and how we store the resulting data for follow-on analysis.

Chapter 4 presents our results and analysis. We explain how data was collected, parsed, and organized and then detail the findings. The focus of this chapter is on what thresholds were able to be identified and the ability of the classifier to identify reverse proxies.

Finally, Chapter 5 gives our conclusions and recommendations on future work. We conclude that an accurate threshold was successfully identified and implemented. Shortfalls are also discussed and suggestions for future work are presented.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2: BACKGROUND AND RELATED WORK

The basis of this research is to leverage the behavior of TCP and the methods used for the connection setup and data transfer of HTTP requests. Therefore, it is important to examine the operation of each of the protocols in order to gain perspective on why and how the timing of each can be differentiated. We begin with a discussion of the protocols and systems we consider in this research. After that we will explore the previous work in this realm that has led to this research. We then explore other attempts to solve similar problems as well as similar methods used to solve other problems.

2.1 Fundamental Knowledge

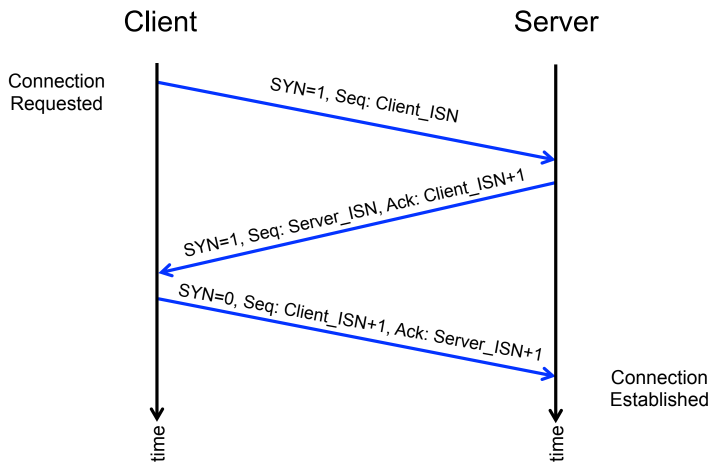
This chapter begins with a brief description of TCP, HTTP, and reverse proxies.

2.1.1 TCP and HTTP

Two transport protocols are provided by the Internet, User Datagram Protocol (UDP) and TCP. The latter is what web clients use to access web page content via HTTP and is standardized in Request For Comments (RFC) 793 [7]. It is a connection- oriented protocol, meaning a client and server must exchange Layer 4 information in order to establish a connection prior to passing higher layer messages such as Hypertext Markup Language (HTML). This connection establishment, or handshake process, is comprised of three steps, aptly named the 3WHS, and shown in Figure 2.1. It is what enables the sockets used by two processes to pass data reliably over computer networks. A TCP connection is considered established after the 3WHS has been completed.

A client sends a TCP segment to a server in the first step of the 3WHS. To ensure a common definition, we use “segment” to refer to the Protocol Data Unit (PDU) at the Transport Layer. The segment is void of Application Layer data and its header has the SYN bit set to 1. This is to let the server know that it is a SYN segment that begins a TCP connection. The SYN segment also contains the initial sequence number from the client.

The second step of the 3WHS occurs when the server receives the SYN segment. It al-



Client
Server
Connection Requested
SYN=1, Seq: Client_ISN
SYN=1, Seq: Server_ISN, Ack: Client_ISN+1
SYN=0, Seq: Client_ISN+1, Ack: Server_ISN+1
Connection Established
time
time

Figure 2.1: TCP 3WHS timing diagram

locates the TCP buffers necessary to handle the connection, records the client's initial sequence number, and assigns its own sequence number. The server then sends a segment back to the client with the client's initial sequence number incremented by one as well as its own initial sequence number. Like the SYN segment, this contains no Application Layer data. It is called the SYNACK segment because it acknowledges the previously received segment by returning the initial sequence number plus one and sets the SYN bit in the header to 1.

Once the client receives the SYNACK segment, it allocates its own TCP buffers and records the server's initial sequence number. It sends a final segment as the third and final step of the 3WHS containing the server's initial sequence number incremented by one to acknowledge the SYNACK segment and sets the SYN bit to 0. This final segment is referred to as the ACK and may or may not contain client-to-server data in the payload [8].

The HTTP is described in RFC 7230 as "a stateless application-level protocol for distributed, collaborative, hypertext information systems" [9] however, Internet users most commonly know this as what brings web page content to their computer monitors. Like TCP, information is passed between the server and client to complete HTTP requests but the PDU at this level is not referred to as a segment; rather, it is called a message.

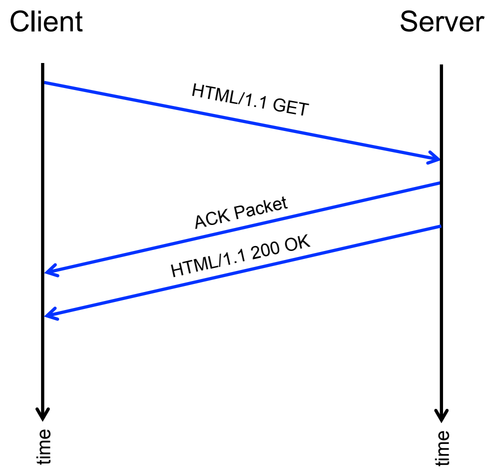


Figure 2.2: HTTP timing diagram

Messages are either requests from clients or responses from servers to the clients. One of the most common types of HTTP client requests is the GET message. This particular request is used to retrieve a representation, or copy, of a resource that resides on a server. Figure 2.2 illustrates that the server acknowledges the GET request first and then follows up with one or more messages in response that contains either a success or error code along with the requested data. The data delivered to the client is called an object and web pages usually contain multiple objects. A web page typically contains text in its base file as well as images and other content such as links and advertisements. Each of these are separate objects and can be delivered by the origin server although, most likely are delivered by other servers that the client has been redirected to via separate TCP connections. For example, the news agency, CNN [10], involves over 170 separate TCP connections just to display its home page.

This simple description of HTTP is the most basic form of request and response and illustrates the fundamental concept of web browsing. Other factors such as caching and intermediaries add complexity and allow for the scalability of HTTP across the Internet, which is one of the primary reasons reverse proxy servers are in use.

2.1.2 Reverse Proxy Servers

As briefly described in Chapter 1, reverse proxy servers take requests from remote clients and gather the data to be returned by maintaining two connections; one to the requesting client as well as another to the back-end server or servers. This enables organizations to remain compliant with regulations such as the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [11] by allowing them to provide content from internal network services to Internet clients without actually storing any of the data in their publicly accessible areas like the Demilitarized Zone (DMZ).

Proxy web servers provide improved performance benefits in the form of workload distribution since one or more servers can handle higher volume “typical'' web requests thereby reducing the load on the origin web server. National Institute of Standards and Technology (NIST) Special Publication 800-41 further explains that many proxy servers have caching enabled in order to speed up web content delivery to clients [12]. The publication also gives evidence of access control applicability for proxy servers in that they can be used for limited firewalling capabilities. A reverse proxy can be used for authentication of clients and even perform “analysis and validation of common application protocols such as HTTP.”

2.2 Previous Work

Our efforts are based largely on previous work done by a former student at the NPS [3]. His research demonstrated a novel method for inferring the presence of reverse proxies by conducting timing analyses from a remote client computer. The basic concept was that a client could determine whether or not a resource provider was positioned behind a reverse proxy by comparing the RTT of the TCP 3WHS to the actual content delivery of the requested web page.

He gathered timing measurements from web requests made to nearly 300 live websites found on Alexa’s top one million websites [13] as well as several more to a list of known malicious sites. This gave him a large set of active websites from which he requested content to measure RTTs. He also added to his resource set by identifying a list of ten alleged reverse proxy servers found by searching Torrent proxy servers in Europe. Despite the large pool of target servers, he only had true knowledge of a single reverse proxy that he had installed in Monterey, California and a single website that was provided by a large internet domain registrar and web hosting business in New Jersey. As is the case with most attempts at inferring Internet topology, his research could not be validated with ground-level truth. He did not have first-hand knowledge about the actual topology of his experimental network devices despite the large resource pool and therefore, the results were largely speculative.

Our primary goal in this research was to expand upon his methodology and apply it to a topology of known clients, servers, and reverse proxies so that we could identify an accurate classifier that would best detect the presence of a reverse proxy server.

2.2.1 Attempts to solve similar problems

As mentioned in Chapter 1, identification and validation of the source of information is a challenge that applies to many aspects of daily transactions. The main difference with computer systems is that the interrogation process must happen nearly instantaneously however, the fundamental concepts remain the same.

Redirection Botnet Seeker (RB-Seeker) was presented at the Network and Distributed System Security (NDSS) Symposium in 2009 as a system aimed toward detecting Redirection Botnets (RBnets). RBnets are sets of infected computers used as an infrastructure to redirect and proxy traffic at the direction of a master computer known as a botmaster. The authors chose to focus their study toward redirection services instead of proxy services because “redirection offers several financial and performance advantages over proxy in terms of content availability, resource utilization, and ease of management” [14]. They supported their argument by explaining that most botents are comprised of commodity home computers and are therefore less capable of reliably maintaining two connections without being detected. Additonally, they are less likely to be available for continuous periods making them more difficult to manage remotely.

Despite the researchers’ aversion to botnets serving as proxies, their methodology could still potentially be adapted toward reverse-proxy detection without timing measurements. Their design involved the use of three subsystems that worked together to detect inconsistencies. One subsystem populated a database of redirections found through links in spam emails and the second populated the same database with network traces using sequential hypothesis testing. The database was later given to the third subsystem which conducted DNS queries and compared them to the data from the other two and made decisions based on hyperplane decision functions.

While their system proved to be very effective in identifying RBnets, it did not incorporate the round trip time measurements that we used. It could be useful to populate a database with timing measurements from different vantage points and then compare them for decision making however, the intent was to create a system that can detect the presence of reverse proxies from only the client's vantage point.

Packet timing measurements were used successfully for the identification of firewalls in a limited-scale research effort conducted by Michigan State University and AT&T Labs [15]. The group measured the processing time of probe packets that they sent through a firewall in order to identify unique packet handling methods that they could then attribute to specific devices. Their research was based on the assumption that a host computer inside the protected network had been compromised and a scanning tool such as nmap [16] had been conspicuously installed and activated. While timing was used to fingerprint firewalls, it required vantage points on both sides of the firewall to identify the packet classification algorithms being used and therefore is not able to be adapted for our research.

Another effort to use time as a metric for identification was by a group of researchers at the University of California, San Diego. The team advanced the concept of remote operating system fingerprinting to that of identifying remote physical devices using very small deviations in clock skew [17]. The authors demonstrated that they could determine whether two devices on the Internet were actually the same regardless of whether they were separated by time, distance, or access technology. This was an advancement in the use of timing measurement because they sought to exploit the deviations in clocks rather than to minimize or eliminate them.

The techniques described in their paper leveraged standardized protocol fields [18] specifically, those in the TCP Timestamps Option and Internet Control Message Protocol (ICMP) Timestamp Requests. In doing so the authors discovered that clock skews are measurable and provide information to fingerprinters (adversaries) regardless of their physical location in the world. They demonstrated that clock skew estimation is independent of topology or access technology in nearly all instances. Furthermore, while these clock skews were constant over time for all tested devices, there was a detectable difference between disparate devices. Their data showed that they could detect at least 6 bits of entropy between unlike devices and that it was enough to determine that they are in fact different. This was a valuable contribution because it showed that only 6 bits can be enough of a partial fingerprint to accurately identify a physical device.

Our research does not apply the same techniques as the preceding research contributions however, they provide motivation that time is a viable method for inferring the presence of devices remotely.

2.2.2 Attempts to use similar solution methods to solve other problems

Our research will rely heavily on the use of PlanetLab in order to collect accurate, relevant timing measurements through the Internet. PlanetLab is an overlay network of over 1,100 machines, or nodes, dispersed internationally at 584 academic, industry, and government sites [19]. It was introduced in 2003 as a research testbed that is fully connected through the Internet thus enabling users to conduct experiments under real-world conditions [20]. There has been significant network-related research conducted using this valuable resource as it provides a relatively low risk experimentation environment through the use of virtual machines from which to test new and innovative programs and methodologies.

In 2008 Colin Dixon, Thomas Anderson, and Arvind Krishnamurthy presented their Denial of Service (DoS) prevention system, Phalanx, and claimed it could counter the effects of botnets with the comparable aggregate power of swarms. The concept was that packets would travel from a sending client, through a random order of participating end-host mailboxes, and eventually be either retrieved or ignored by their intended destination. The random ordering of mailboxes that packets traversed would allegedly foil DoS attacks by even the largest botnets because there would be no known routes or nodes for them to target. They demonstrated that their system could in fact withstand a multimillion-node botnet and reasonably be deployed by a single large Internet Service Provider (ISP) by using PlanetLab.

They first assigned two PlanetLab nodes as end clients (one in Berlin, Germany and the other in North Carolina) and ten mailbox nodes between them in order to compare latency measurements of normal UDP traffic and Phalanx protected traffic. They then simulated an attack on half of the mailbox nodes by intentionally dropping 75% of the packets at the nodes under attack. Their next phase focused on the scalability of their system so they needed a means to test it in a large-scale environment. Since PlanetLab is not nearly large enough to replicate Internet-scale testing by itself, the researchers modified the nodes' roles. They created a simulator based on a probe of 7,200 actual nodes known to be part of a major Content Distribution Network (CDN). The simulator then modeled the nodes to be participating Phalanx mailboxes and then re-assigned the original PlanetLab nodes to simulate web servers under attack. The final phase of their methodology was to test their system at a multi-million bot level. The researchers again used the simulated mailboxes but increased the access link bandwidth to reflect that of a swarm at a similar scale [21]. The significance of this research effort beyond the successful demonstration of Phalanx, was the effective use of PlanetLab to showcase the prototype as well as enhance an Internet-scale simulation. The network provided the means to simulate a DoS attack without having to deploy a botnet in the wild and allowed for the simulation of actual infrastructure all without interfering with real Internet traffic.

Mapping all or parts of the Internet is another challenge many researchers and private companies work hard to solve. Approaches known to gain knowledge about network topologies, operating systems, and specific middle-box devices have proven successful. There have even been some methods developed to identify the presence of, and even fingerprint reverse proxy servers via remote means. One such method involved using Extensible Markup Language (XML) for fingerprinting [22] but the primary method our research will employ is timing analysis.

CHAPTER 3: METHODOLOGY

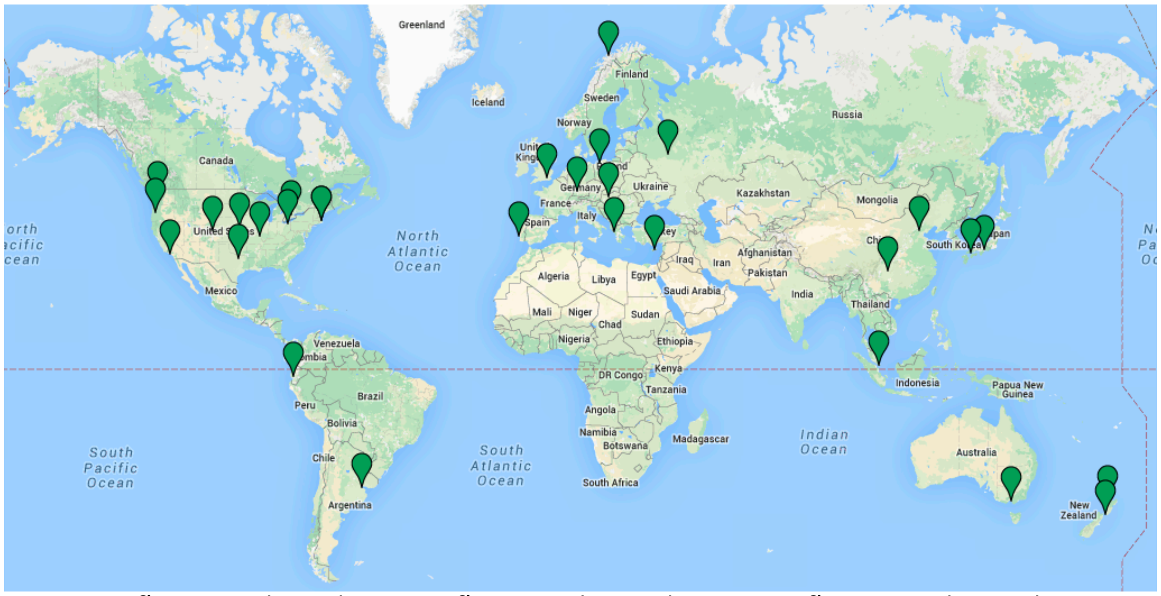
The conceptual foundation of this thesis was derived from the findings of the previous research [3]. The first step of the methodology was to repeat the experiment on a known set of machines. To accomplish this first step we established several PlanetLab nodes worldwide and conducted our experiment on a widely dispersed set of clients, web servers, and reverse proxies. Our methodology was designed to capture and measure the RTTs of TCP packets from the client machine's perspective. From these captures, we calculated the ratio of two particular RTTs and used it as the identifying attribute of the known configuration. The attribute ratio and the ground truth of the configuration were then used to find a classifier that could determine the presence of a reverse proxy in a separate captured flow.

3.1 Data Collection

We began our study by instantiating a slice of PlanetLab nodes representative of the globally distributed nature of the Internet. We used thirty nodes as shown in Figure 3.1; five in Asia, three in Australia/New Zealand, nine in Europe, eleven in North America, and two in South America. Some nodes were located in close proximity to each other for example, two nodes in Boston, Massachusetts, while others were separated by great distances like the nodes in Norway and New Zealand. Our test methodology also included instances where the proxy and web server resided on the same machine to represent the smallest possible distance between two points. There were other nodes available that would have provided for even greater geographic separation however, many that participate in the PlanetLab consortium were not reliable during our testing period so were not included in this set.

The high-level concept of our measurement configurations is shown in Figure 3.2. A client on a node made one of three types of web requests; directly from the web server, via a remote reverse proxy server, or via a reverse proxy server on the same machine as the web server.

Each node served as the client, the reverse proxy, and the web server depending on the permutation being executed so that every iteration was conducted under the same distance

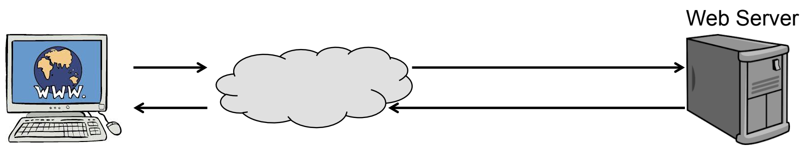


City	Lat	Long	City	Lat	Long	City	Lat	Long
Nagoya, Japan	35.1566	136.925	Moscow, Russia	55.75	37.6	Cleveland, OH	41.5022	-81.6079
Chongqing, China	29.5372	106.602	Lisboa, Portugal	38.45	-9.09	Lincoln, NE	40.8	-96.6667
Beijing, China	39.9606	116.359	Nicosia, Cyprus	35.15	33.36	Los Angeles, CA	33.9815	-118.46
Singapore, Singapore	1.37	103.8	London, England	51.498	-0.176655	Boston, MA	42.37	-71.03
Hiroshima, Japan	34.4403	132.415	Budapest, Hungary	47.4725	19.06	Cambridge, MA	42.363	-71.0926
Hamilton, New Zealand	-41.2902	174.768	Ioannana, Greece	39.6182	20.8386	Waterloo, Canada	43.4726	-80.5422
Auckland, New Zealand	-37.7892	175.318	Tromso, Norway	69.6813	18.977	St. Louis, MO	38.6479	-90.3015
Melbourne, Australia	-37.9096	145.133	Stuttgart, Germany	48.7835	9.17517	Boulder, CO	40.0072	-105.262
Guayaquil, Ecuador	-2.15	-79.96	Koszalin, Poland	54.2047	16.1972	Seattle, Washington	47.6531	-122.313
Buenos Aires, Argentina	-34.5975	-58.3985				Arlington, TX	32.73	-97.115
						Eugene, OR	44.04	-123.06

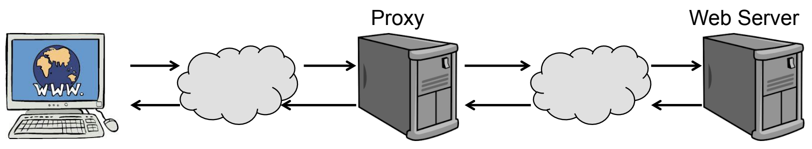
Figure 3.1: Map of PlanetLab nodes used. Each of the green pins on the map [23] corresponds to the location of one or more PlanetLab nodes. The table below the map lists the city and coordinates of each of the nodes

and hardware conditions. This maximized the utility of the PlanetLab nodes to make most efficient use of the data collection period and placed similar loads on each machine. The intent was to increase the web traffic to our machines thus, more closely replicating what is assumed to be the load on a highly-visited production web server. Our set of thirty nodes also provided the opportunity to gather a large data set for analysis.

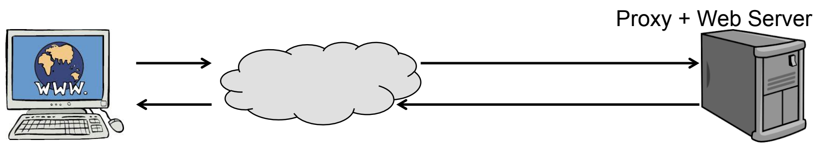
We tested every permutation of realistic web service configurations. We define realistic configurations by first describing unrealistic ones. It would be unrealistic for a web client, for example, to request a web page from a web server that happens to also be on the same machine as itself. If this were the case, it would be trivial to determine whether or not the web server was behind a reverse proxy because the operator has access to the entire



(a) Direct connections



(b) Remote reverse proxy



(c) Local reverse proxy

Figure 3.2: The three different instances of timing measurement configurations

machine. The realistic configurations we examined are described in more detail in the following paragraphs.

We began our measurements with a single iteration of direct connections between the clients and web servers. Every node requested the web page from every other server 250 times as shown in the pseudo-code presented in Algorithm 1. All 30 nodes opened a text file that contained the IP addresses of the other 29 nodes and requested the web page from the server that was active on each respective node. The scale of the directly connected permutations can be represented by $(n(n-1)) \times 250$ where $n =$ number of nodes in slice.

This comes to 217,500 direct connection flows to analyze.

Algorithm 1: Method to directly request the web page from every web server

```
for IP in list of PlanetLab nodes do
    start tcpdump;
    wait 2 seconds;                 /* to allow tcpdump to start capturing */
    for i = 1; i <= 250; i++ do
        wget IP;                        /* save results to local pcap file */
        traceroute every 25th IP;        /* save results to local txt file */
    end
    killall tcpdump
end
```

The scale of measurements increased significantly when the reverse proxies were introduced. For each iteration, only one node served as the web server, s. Each client c requested the web page via every reverse proxy p except when c and p were the same machine because it is not realistic for a client to use itself as a reverse proxy. Likewise, c was never equal to s for the same reason. It is acceptable however, that p and s could be the same node so we incorporated that configuration into our methodology. Pseudo code is presented in Algorithm 2 to help clarify the method used to test every permutation of our PlanetLab set. Figure 3.3 shows a visual representation and description of the necessary connections for a single iteration with only three nodes. Using the formula $n((n-1)(n-1))$ we calculated the number of iterations needed to measure every permutation of our slice which was 25,230. Again, every request was made 250 times resulting in 6,307,500 reverse proxy flows to collect.

Algorithm 2: Method to request the web page from every reverse proxy

```
for Server IP in list of PlanetLab nodes do
    for Proxy IP in list of PlanetLab nodes do
        Destination IP = Server IP;            /* Point proxies to web server */
        /* All nodes execute from here except the target web server   */
        for IP in list of PlanetLab nodes do
            start tcpdump;
            wait 2 seconds;           /* to allow tcpdump to start capturing */
            for i = 1; i <= 250; i++ do
                wget IP;                    /* save results to local pcap file */
                traceroute every 25th IP;    /* save results to local txt file */
            end
            killall tcpdump
        end
    end
end
```

In addition to controlling the traffic conditions, we used what we assumed to be similar hardware as well as installed identical software on each node to minimize variables that could impact our timing measurements.

The actual hardware configurations of the PlanetLab nodes could not be absolutely determined however, they all ran virtual instances of the Fedora Operating System. The majority of the nodes in Europe used Version 14 (Laughlin) while the rest of the nodes used Version 8 (Werewolf). Every node in our set used the Cubic implementation of TCP with TCP Segmentation Offload (TSO) enabled and set to the default setting of 3. TSO is a method of offloading work from the Central Processing Unit (CPU) to the Network Interface Card (NIC) thereby increasing the throughput for outgoing data. When TSO is enabled, large chunks of data are sent to the NIC which then forms the TCP segments. When disabled, this work is done entirely by the CPU. The TSO setting determines how much of the TCP congestion window can be filled by one of these chunks. The settings

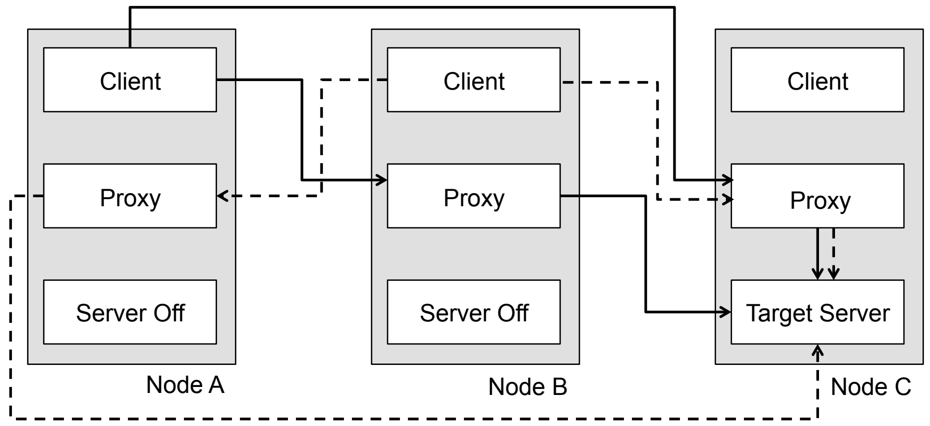


Figure 3.3: Single iteration with three nodes $(n = 3)$ showing number of web requests is $(n-1)(n-1)$ when Node C is the target web server. Node A requests HTTP content from Node C via the proxies at Nodes B and C. Node B requests HTTP content from Node C via the proxies at Nodes A and C. Node C does not participate in this iteration as it is not normal for clients to request web content from itself. For all possible permutations in this diagram there would need to be $n((n-1)(n-1))$ or 12 total iterations to allow for each node to be the target server.

range from 2 to 16 with the lower numbers resulting in larger chunks and less work for the CPU [24].

We installed the same software on all nodes to minimize the variables that could impact our timing measurements. Apache Hypertext Transfer Protocol Server (httpd version 2.2) was installed as the web server. The web page was designed to be 1KB in size so that it would be delivered in a single data packet. Normally, web pages are larger than what can be sent in a single IP packet and must be sent over several packets. We were only interested in the RTT of the first data packet so the additional data was inconsequential to our measurements. Additionally, the standard port number for HTTP traffic, port 80, was not available for our use on the PlanetLab nodes because the resources had to be shared among all users. To account for this we assigned a non-standard port for the Apache web server to listen on.

We installed NGINX (0.6.33) [25] as the reverse proxy on each node even though there were other tools available such as Squid [26] and Apache [27] that are widely used and handle server-side connections differently. The fundamental property of requiring more time to retrieve and deliver content than to connect to the client holds true despite the software used [3] however, so we did not collect data using these other applications. Again, we used a non-standard port number to listen on since we could not use port 80.

Finally, the standard tools *tcpdump* [28] and *wget* [29] that are included in the Fedora distributions of PlanetLab were used as the client tools to request web pages and record packet data. We filtered our *tcpdump* to capture only packets destined for or coming from our set of nodes and their non-standard port numbers to reduce the size of the packet capture files down to only what was pertinent to the investigation.

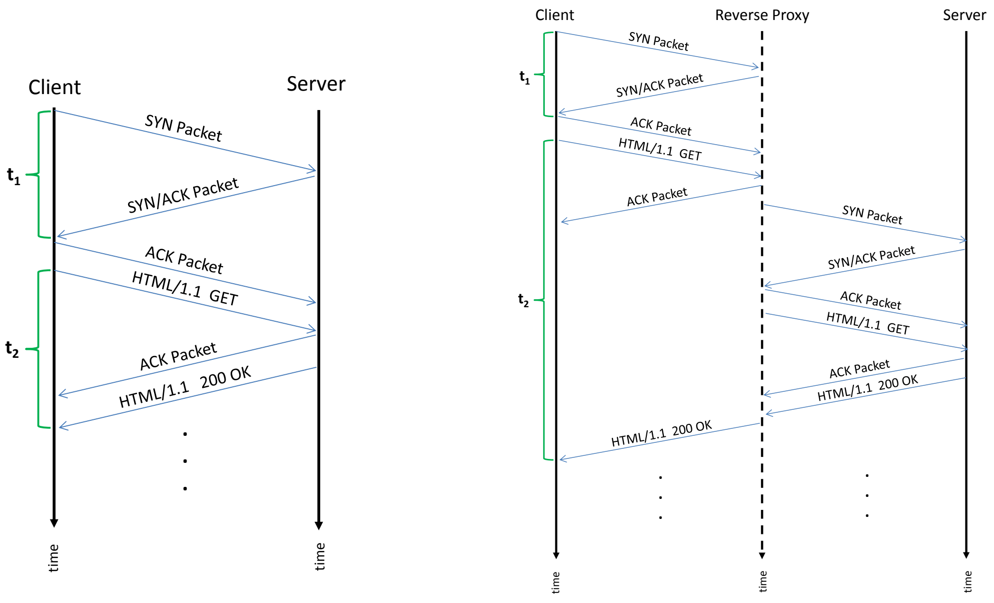
We used the previous research methods and tools [3] to accurately measure the RTTs of the TCP 3WHS and HTTP request. The timing measurements were taken based on two RTTs as shown in Figure 3.4. The first RTT was the elapsed time from when the SYN packet was sent by the requesting client until the corresponding SYN/ACK packet was received. This time was annotated as t_1. The second RTT, t_2, was calculated from when the HTML GET request was sent to when the first data packet, the HTML *200 OK* message, was received.

A script was used to automate the initiation and recording of the timing measurements. The packet analyzer, *tcpdump*, was started first to begin capturing packets. The script paused for two seconds to allow the analyzer to initialize and then we used *wget* to retrieve the web page 250 times in succession.

We configured *wget* to disable server-side caching so that every request would reach all the way to the web server and prevent inaccurate measurements due to attempted performance enhancements at the reverse proxy. The feature was disabled by using the "–no-cache" HTTP option in the *wget* command which sends the remote server instructions to get the content from the remote server and not return a cached version. The *wget* documentation explains that it "is especially useful for retrieving and flushing out-of-date documents on proxy servers." All nodes in the set executed the script in parallel and saved the packet captures to local pcap files as well as the traceroute data to local text files.

The order of web servers that content was requested from was different on each client so that the overall traffic was evenly distributed. This prevented all requests going to a single web server simultaneously and helped alleviate potential weather effects or time-of-day phenomena that is known to impact traffic patterns [30] [31].

Once each iteration was complete, we used another script to retrieve the packet capture files



(a) Direct connection (b) Via reverse proxy

Figure 3.4: Side by side comparison of web request by direct connection and via reverse proxy from [3]

from each node and saved them to a central machine for later analysis. We then deleted the files from each node in preparation for subsequent iterations.

3.2 Data Analysis

After all the data had been captured, we divided it into two sets: a training set and a test set. The training set was used to evaluate and determine the best classifier that would detect the presence of a reverse proxy server. We did not initially subdivide the proxy data to isolate the proxy servers that were co-located with the web servers and those that were not. Our objective was to determine the best classifier that would decide if there was a reverse proxy and not how far away it was. Later however, we did find it useful to separate the two proxy cases and explored the results further.

The classifier was then applied to the test set to determine its accuracy on the set. Again, our classifier was based on the ratio t_2 to t_1. We anticipated that a ratio approximately equal to 1 reflected a directly connected client and web server while anything greater meant that there was a reverse proxy in the path. The goal, then, was to find the most precise ratio that would give the best accuracy in predicting whether our traffic was coming from a proxy.

The training set was comprised of a randomly selected 80% of the total data captured. We used the tools created in prior research [3] to parse the packet capture files and extract the pertinent information. We made minor modifications to account for the non-standard ports that were recorded. We also added features to pre-calculate the ratios and record the actual configuration of each flow. A configuration setting of *0* meant that the client was receiving the content directly from the web server while a setting of *1* meant that it was not.

Once the training set of data was compiled we used the open source machine learning and data mining software suite, Orange (2.7.8.dev) [32], to find the ratio that best determined web server configuration. Since the GET/3WHS ratios were already calculated we only needed to pass them and the binary configuration values to Orange. Orange then used decision tree learning to produce a classification tree based on the configuration as the class and the ratio as the attribute of the class. We took the ratio at the trunk of the classification tree and used it as our classifier.

The classifier was then used on each flow in the test set to decide whether or not it was a web server behind a reverse proxy. The decisions were recorded in a confusion matrix (Table 3.1) from which the performance of the classifier was evaluated. Each decision was recorded as d_{ij} where i is the actual configuration and j is the predicted configuration. For example, an entry of d_{01} means that the classifier predicted the web server was behind a reverse proxy ($configuration = 1$) when in actuality is was not ($configuration = 0$).

Finally, the overall performance of the classifier was calculated in terms of its accuracy, precision, negative predictive value, recall, specificity, and F1 score. These measures are discussed in Chapter 4.

Table 3.1: Confusion matrix template

		Prediction	
		$R.Proxy = 1$	$Direct = 0$
Truth	$R.Proxy = 1$	d_{11}	d_{10}
	$Direct = 0$	d_{01}	d_{00}

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4: RESULTS AND ANALYSIS

The execution of the test methodology required 50 days to complete from December 1, 2014 to January 19, 2015. Challenges in PlanetLab availability were the primary factor in the lengthy data collection period however, like actual systems in the Internet, it only added to the realism of actual Internet performance. We were successful in collecting a large data set and were able to find an accurate classifier ratio to determine the presence of a reverse proxy.

4.1 Data Set Acquired

The data collection plan was designed to capture 6,525,000 flows in the best-case scenario. This upper bound would be feasible if there were no incomplete 3WHSs, HTTP requests, or hardware failures. The performance of PlanetLab was such that we ultimately captured 6,225,044 total flows, or 95% of the best potential outcome (Table 4.1). The low rate of capture in direct flows was a result of node restarts or failures during the measurement period. The measurement period for the configuration without proxy servers was much longer in duration since all 30 nodes were executing 250 web requests to the other 29 nodes in parallel.

The average duration of the non-proxy configuration measurement period was seven hours and on none of our three attempts did all 30 nodes stay online the entire time. By comparison, the duration of each of the proxy configuration measurement iterations was three to four hours. The difference in time is due to two factors. First, the direct connection configuration sent over 7,000 more web requests than a single proxy iteration due to the realistic configuration rule explained in Chapter 3. Where only one scenario was ruled out in the direct configuration, a client requesting content from itself, two scenarios were ruled out in the proxy configuration. A client would not request content from itself as a web server and likewise, it would not request content through itself as a reverse proxy. The exclusion of the additional node leads to the second reason for the difference in collection periods. The direct configuration required that all web servers be operational during the collection period while in the proxy configuration, only the target web server needed to remain operational. While it is true that all the nodes were operating as reverse proxies and needed to remain online, we found that the longer the collection period lasted, the more likely a node would restart or fail.

We examined several files to find the cause of the unsuccessful captures and found that they were indeed due to incomplete flows resulting from timeout thresholds being exceeded and nodes going offline. We also found that the majority of the unsuccessful flows were attributed to five specific PlanetLab nodes (Boulder, CO; Los Angeles, CA; Lisbon, Portugal; Seattle, WA; and Chongqing, China).

The unsuccessful flows were a reflection of some of the challenges we experienced in data collection. One of the first difficulties encountered was in establishing the most reliable set of nodes from our slice of nodes to use for experimentation. We found that it was not the case that the nodes adhered to any standard base operating system version or permission sets. There were many nodes that would not allow any programs to bind to non-standard ports which prevented our web servers and reverse proxies from operation. Others would connect to various Fedora [33] repositories and install programs of different builds for the varying PlanetLab kernels. This did not prevent the installation or operation of the web servers or reverse proxies but did require us to prepare separate configuration files depending on the kernel version and therefore, slowed down the parallel modification to those configuration files.

Another challenge was that PlanetLab nodes regularly restarted unexpectedly, presumably for maintenance. One node in particular, at the University of Texas, Arlington, would routinely restart every night. Hardware failure events also occurred on more than one occasion. Most times the faults were resolved within a few days; however, one node was inoperable for nearly two weeks and required significant effort just to find a point of contact for the node.

Table 4.1: Summary of flows captured

Type of Flow	Number Expected	Number Captured	Percent Captured
No Proxy	217,500	193,859	89%
With Reverse Proxy	6,307,500	6,031,185	96%

One unanticipated challenge resulted from our efforts to parallelize the test execution. All nodes in the set were simultaneously requesting web pages as well as serving as reverse proxies during the measurement iterations. Each node that was requesting web pages in a client role was capturing the packets and saving them in *pcap* files. On multiple occasions, two nodes would request and serve web pages from and to each other at the same time which would cause the flow to be recorded from both vantage points. We discovered multiple instances of impossible RTT ratios during the analysis phase of the experiment which led to further investigation. Our method to parse flows from the captured *pcap* files relied on port numbers to determine the start and subsequent states of each flow. This resulted in several flows being extracted from the *pcap* files and recorded as flows from the wrong perspective. The outcome was that the HTTP content request and delivery RTTs were several orders of magnitude shorter than the 3WHS RTTs. These extraneous "backward" flows were easily identified and removed prior to being included in the training and testing sets.

We plotted ratios for each of the three configuration types once the erroneous and incomplete flows were removed from the data set. The ratios were all approximately equal to 1 in the instances when a client was directly accessing the web server (Figure 4.1). This indicated that the overwhelming majority of the 3WHS RTTs and HTTP RTTs were nearly identical since both processes involved only the requesting client and web server, as well as the intermediate topology of Internet devices. The results also confirmed that there was a noticeable difference in ratios when a client requested a web page from a web server behind a reverse proxy and one that was not. This demonstrated that the additional time required for the back- end processing was indeed detectable from a client machine's perspective.

The ratios were higher in the two non-direct configurations because the reverse proxy had to connect to the web server, retrieve the web content, and then deliver it to the requesting client. The instances when the reverse proxy was on the same machine as the web server resulted in a smaller variation in ratios than those when the reverse proxy was remotely located. An interesting observation is that the ratios for reverse proxies on the same machine as the web server providing the content resembled the results of those that were remotely connected. One could assume that since distance was not a factor that the ratios would more closely resemble the ratios of clients requesting content directly from the web server. We will explore these results further at the end of this chapter.

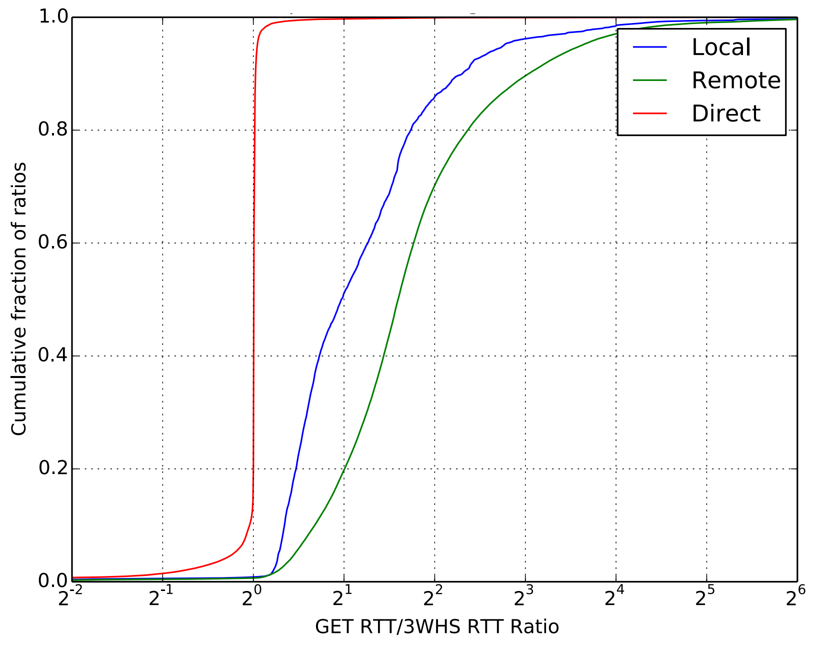


Figure 4.1: Comparison of GET/3WHS ratios. These CDFs illustrate the difference in ratios from clients retrieving content directly from the web server or via a reverse proxy that is either directly or remotely connected to the web server.

The CDFs reinforced the hypothesis that timing analysis alone can detect the presence of a reverse proxy and supported further exploration to find the most accurate classifier.

The data sets were divided into training and testing sets for use in machine learning and testing. The training set was comprised of a randomly selected 80% of the proxy flows concatenated with a randomly selected 80% of the direct flows in a single table. Figure 4.2 illustrates the division of the data sets. Likewise, the testing set consisted of the remaining 20% of the proxy flows concatenated with the remaining 20% of the direct flows in a single table. Table 4.2 contains sample data from one of the data sets built following the combination of proxy and non-proxy flows. This method was adjusted later to enable cross-validation as described in Section 4.4. The first column of the table identifies the client port number which is how we uniquely identified each flow. The next two columns display the client (source) IP address and the IP address of either the web server in the case of a direct connection, or the reverse proxy in a proxied connection. Columns 4 and 6 are the actual

RTTs of either the 3WHS or the GET. The column in between lists the number of flows captured based on the client port number. This was useful in identifying flow instances where a port number collision had occurred. In the very few cases that this happened, we only considered the first captured flow to prevent any ambiguity in flow timing. The seventh column lists the calculated ratio for each flow and the last column represents the ground truth value of the actual configuration. A value of 0 means that the client directly accessed the web server while a value of 1 means that the connection went through a reverse proxy.

Table 4.2: Sample of data in table of captured flows

Client Port	Src_IP	Dst_IP	3WHS	Flows	GET	Ratio	Truth
43520	128.138.207.54	128.208.4.199	0.042830	1	0.043359	1.012351	0
55264	128.208.4.199	194.42.17.121	0.215998	1	0.216253	1.001181	0
39261	128.223.8.114	195.130.121.205	0.242246	1	0.243015	1.003174	0
44023	62.108.171.76	211.68.70.40	0.202333	1	0.910982	4.502390	1
56002	211.68.70.40	194.42.17.121	0.232182	1	0.650417	2.801323	1

4.2 Initial Training Set Results

As mentioned earlier, we used Orange as the machine learning tool to calculate the best classifier ratio. The program built a classification tree using the actual configuration, proxy or direct, as the class and used the ratio of each flow as the attribute associated to the class. Once Orange completed the decision tree it returned a threshold of 1.044. This indicated that the time required for the client to receive the first data packet of a requested web page from a target server was nearly identical to the time required for the initial TCP 3WHS.

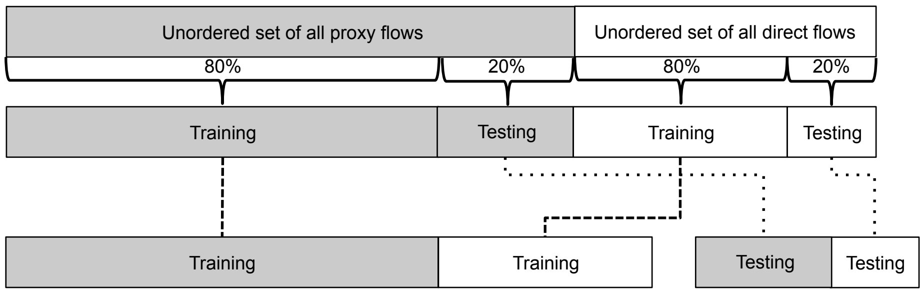


Figure 4.2: Initial division of the data into training and testing sets. The full data set of flows was divided into training and testing subsets consisting of 80% and 20% of the flows, respectively. A classification tree was built using the combined training subset and the resulting classifier was evaluated using the combined testing subset.

4.3 Performance of Initial Classifier on Test Set

After we identified the potential classifier, we evaluated it using the remaining 20% of data that was set aside as the test set. We wrote a short Python [34] program to predict the presence of a reverse proxy based upon the ratio. If it was greater than the classifier, it was predicted to be a reverse proxy connection. If it was less than or equal to the classifier, it was predicted to be a direct connection. We then compared the predicted class to the actual class and annotated whether it had correctly identified it or not in a confusion matrix.

For testing purposes we needed to define the type of correctness or incorrectness that the classifier had predicted. The completed confusion matrix is shown in Table 4.3 and reflects the results of our testing.

Table 4.3: Confusion matrix of classifier on test set

		Prediction	
		R.Proxy	*Direct*
Truth	*R.Proxy*	1,198,633	6,896
	Direct	1,503	37,150

We then used the data in the confusion matrix to calculate the quality of the classifier in determining the presence of a reverse proxy server. For this, we considered accuracy, precision, and recall.

The accuracy reflects whether the test actually measured what it was supposed to. It gives a general understanding of the performance of our classifier and was calculated as:

$$\begin{aligned} Accuracy &= \frac{Number\_of\_correct\_predictions}{Total\_number\_of\_predictions} \\ &= \frac{1,198,633+37,150}{1,244,182} \\ &= \frac{1,235,783}{1,244,182} \\ &= .99 \end{aligned} \tag{4.1}$$

Although this quick calculation shows a very high accuracy, it does not expose which predictions were most successful between the direct and proxy connections. It was necessary, then, to examine the results more closely.

Precision, or Positive Predictive Value (PPV), is the probability that a reverse proxy was present when the classifier predicted that there was one. In other words, given a prediction of "proxy" from the classifier, how likely was it to be correct? The Negative Predictive Value (NPV) is the inverse of precision. It reflects the probability that direct connections were truly direct connections when the classifier determined that no reverse proxy was in the path.

$$
\begin{aligned}
Precision &= \frac{TP}{TP+FP} \\
Precision &= \frac{1,198,633}{1,198,633+1,503} = .99 \\
\text{NPV} &= \frac{TN}{TN+FN} \\
\text{NPV} &= \frac{37,150}{37,150+6,896} = .84
\end{aligned}
\tag{4.2}
$$

The results show that when the model predicted that the configuration was with a reverse proxy that it was correct 99% of the time while it was only correct 84% of the time with direct connection.

Recall, also known as sensitivity, is the measure of the ability for a prediction model to select instances of a certain configuration from a data set and corresponds to the True Positive (TP) rate. In the context of this research, a positive prediction is one that predicts a reverse proxy connection. When the recall is high, there is a greater probability that the test detects a reverse proxy when it is indeed present in the path. The inverse of recall is specificity, which corresponds to the True Negative (TN) rate or the probability that the path is predicted to be without a proxy when the proxy is absent. The question we sought to answer here was, given a configuration, what was the likelihood the classifier would detect it?

$$
\begin{aligned}
Recall &= \frac{TP}{TP+FN} \\
Recall &= \frac{1,198,633}{1,198,633+6,896} = .99 \\
Specificity &= \frac{TN}{TN+FP} \\
Specificity &= \frac{37,150}{37,150+1,503} = .96
\end{aligned}
\tag{4.3}
$$

Here, we see that when our model was presented with a RTT ratio it was able to detect reverse proxies 99% of the time and direct connections 96% of the time.

Finally, we considered the F1-Score or, the overall measure of the classifier's performance when predicting the presence of a reverse proxy server.

$$\begin{aligned} F1 &= 2 \times \frac{Precision \times Recall}{Precision + Recall} \\ F1 &= 2 \times \frac{.99 \times .99}{.99 + .99} = 2 \times \frac{.98}{1.98} = 2 \times .49 = .99 \end{aligned} \tag{4.4}$$

The results show that a classifier of 1.044 seemed accurate when predicting that there was a reverse proxy in the path between a client and web server. The low NPV however, indicated that it may not be reliable in confirming the lack of one and therefore, made our classifier less trustworthy.

We then considered the possibility that the initial split of the data set into training and testing subsets was less than optimal for generating and evaluating a classifier. To address this we conducted a 5-fold cross validation on the entire set of flows using a classification tree for each fold in Orange. The same classifier of 1.044 was determined by the learning algorithm and the results were nearly identical to the analysis that we conducted previously as shown in Table 4.4.

Table 4.4: Comparison of evaluation of classifier on initial 80/20 split and after 5-fold cross validation with classification tree learning

	Initial 80/20	5-Fold
Accuracy	.99	.99
Precision	.99	1.0
NPV	.84	.84
Recall	.99	.99
Specificity	.96	.97
F1	.99	1.0

The results support the indication that the classifier was not as reliable in determining direct connections as it was in finding reverse proxy connections.

4.4 Adjusting the Methodology

The analysis of the initial test set performance revealed a shortcoming in the overall ability of the classifier to detect a directly connected server. A cursory study of the data implied sample bias in that there were 30 times more proxy samples collected than the non-proxy samples. To address this, we divided up the proxy sample training set to approximately match the directly connected training set and re-calculated the results. Since the 5-fold cross-validation resulted in the finding that the initial 80% training/20% testing split of data was sufficient we reused the same training and testing subsets here.

The proxy training set was divided into randomly selected subsets of 155,666 flows to be approximately the same size as the full non-proxy training set of 155,206 flows. The flows were selected at random and without replacement to fully distribute all flows. We concatenated each subset with the direct set (Figure 4.3) and recalculated the classification tree for each pair. After we completed the calculations, we averaged the results to find a new classifier ratio of 1.134387097.

We divided the test set of proxied flows in the same manner and finished with subsets of 38,888 proxy flows each concatenated to the set of 38,653 testing direct flows. We then evaluated the classifier against each test subset and filled in a confusion matrix for each. From that, we calculated the performances and then averaged the results to find the figures in Table 4.5 under the column for the initial classifier results.

The results show that by using the new classifier ratio, all performance categories, including NPV, were 99% correct.

We again incorporated methods from Orange on the entire data set to build confidence that there was not just an optimal split of training and testing flows. Our program randomly, and with replacement, selected a matching number of flows from the proxy set as was in the full direct set. It then ran a 5-fold cross validation using a classification tree learner to generate the candidate classifier of 1.134, evaluate it, and report the results. We executed the program in a loop over 100 iterations and then calculated the average. Although the new classifier of 1.134 was only calculated to the nearest thousandths, it performed the same, as shown in the third column of Table 4.5. The measures are only shown to two decimal places in this case because of a limitation in the way Orange methods returned calculations.

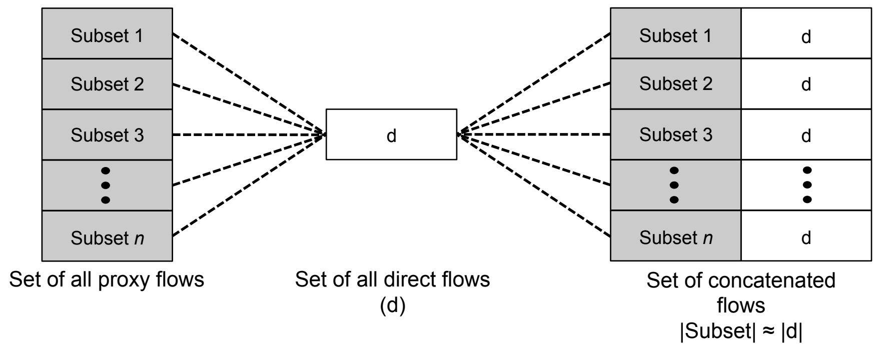


Figure 4.3: Modified subset method on data set. The subset of proxy flows that was separated for training was divided into further subsets of approximately the same size as the set of direct flows that was separated for training. Each of these proxy sub-training subsets was concatenated with the training subset of direct flows and saved as a combined set. The same method was used to subdivide and concatenate the testing subsets for evaluation.

Table 4.5: Final results of classifier performances with sample bias removed

Measure	Initial 1.134387097	Final 1.134
Accuracy	0.987997614	0.99
Precision	0.987616051	0.99
NPV	0.988382302	0.99
Recall	0.988462326	0.99
Specificity	0.987530075	0.99
$F1$	0.988038971	0.99

4.5 Evaluating the Strength of the Classifier

Our results appeared to be very promising; however, we needed to evaluate the classifier to see how increasing or decreasing it would affect the outcome. This will be useful for those wishing to adjust which areas can be subject to higher risk. For example, it could be the case that missing some instances of connections with proxies is acceptable as long as confidence can be increased that those proxies that were identified were actually true. To accomplish this we adjusted the classifier 10% both higher and lower then evaluated it against our data set. Adjusting the optimal classifier of 1.134 by 10% resulted in a new lower classifier of 1.0206 and a new higher classifier of 1.2474.

4.5.1 Evaluating the New Classifier Against the Full Data Set

The first division of data was into two categories, proxy and direct, to determine how adjusting the classifier would impact the detection of a reverse proxy. We randomly selected an equal number of flows from the proxy set and concatenated them to the set of direct flows. The resulting table was then equal parts of proxy and direct flows. The three classifiers (lower, optimal, and higher) were used to evaluate the data in the table and the results stored in a confusion matrix. We replaced the randomly selected proxy flows and then repeated the process with a new randomly selected set of proxy flows concatenated with the set of direct flows. This process was repeated ten times and the results averaged to produce a final confusion matrix. Finally, we calculated the measurements and plotted them to a graph shown in Figure 4.4.

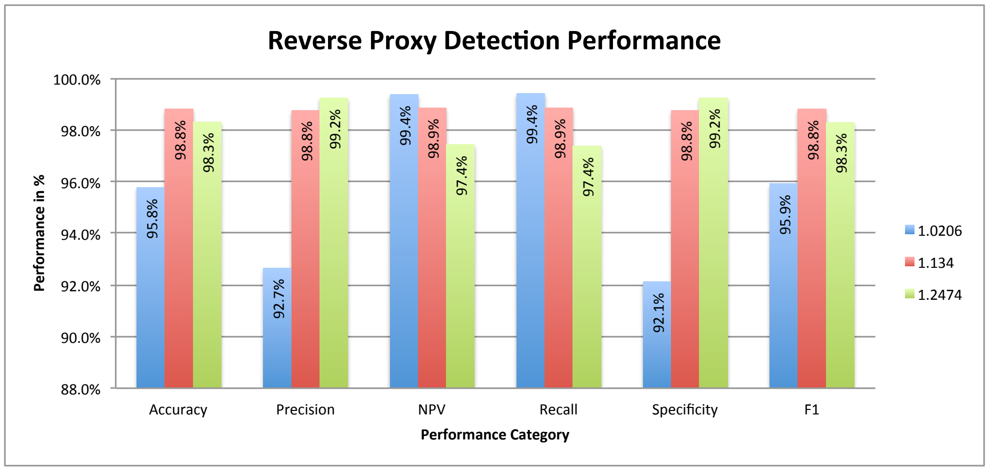


Figure 4.4: Graph of classifier performance when adjusted by 10% higher and lower. The lower (1.0206), optimal (1.134), and higher (1.2474) classifiers' performance measures are compared against each other.

The results show that the optimal classifier of 1.134 had a better general performance than when the classifier was adjusted. Decreasing the classifier resulted in better recall and NPV however, performed worse in all other categories with the most significant drop in precision and specificity. Therefore, by lowering the classifier we were able to better detect a reverse proxy at the expense of higher False Positives (FPs).

We observed better performance in precision and specificity when the classifier was increased. Although the performance declined in all other categories, the change was less significant than when the classifier was decreased. Increasing the classifier by 10% allowed for a slightly better probability that a reverse proxy was present when the classifier predicted that it was. This improvement however, caused the classifier to miss instances of reverse proxies in the data set.

Our results show that the classifier ratio can be tailored to meet an organization's goals. If the objective is to find the highest quantity of reverse proxy servers with the understanding that some will be falsely identified, then the classifier should be lowered. If on the other hand, the objective is to increase confidence that detected reverse proxies are correct, then the classifier should be raised. Figure 4.5 illustrates an example of these findings by showing where the classifier threshold should be set to eliminate all instances of direct, or non-proxy connections. It also reflects that by doing so, the precision is increased at the expense of a higher FP rate.

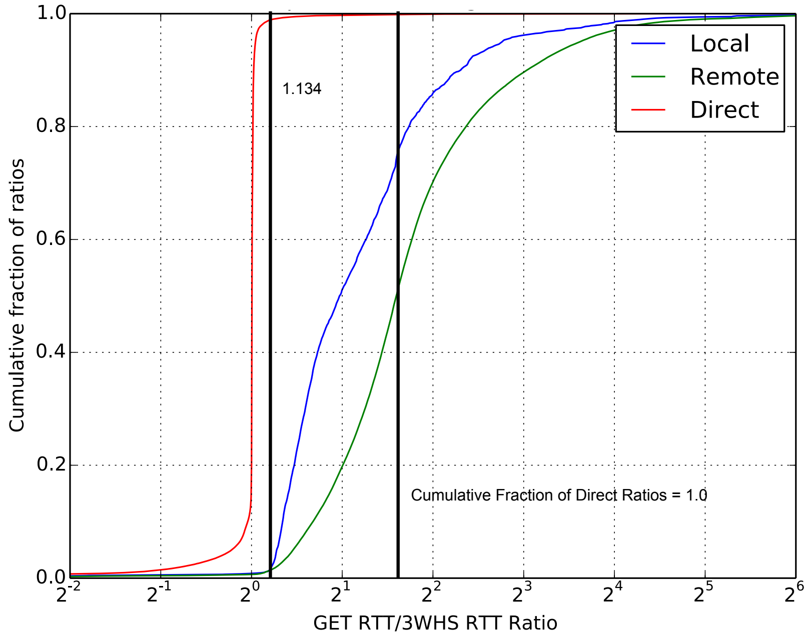


Figure 4.5: The two black lines on this figure represent the optimal classifier of 1.134 (left) and where the threshold must be adjusted to so that the full set of reverse proxy detections includes no instances of direct configurations (right).

4.5.2 Evaluating the New Classifier Against Local and Remote Proxy Configurations

The final area we sought to explore was whether adjusting the classifier had more impact on connections where the reverse proxy was co-located with the web server or remotely connected. We divided the proxy set of flows into local and remote flows. We then performed the same random sampling and concatenation with the direct set as we did when evaluating the adjusted classifiers on the full data set. We first evaluated the adjusted classifiers against the local reverse proxies (Figure 4.6) and then against the remote reverse proxies (Figure 4.7).

Decreasing the classifier did not cause a significant performance change to either the local or remote proxy configurations when compared to the evaluation against the full combined set. The greatest difference was in the F1 score where the performance on the local reverse proxies only dropped by 0.4%, from 95.9% in the full set to 95.5% in the local set. Increasing threshold however, impacted the performance of the classifier on the set of local reverse proxies much more significantly in all categories except precision and specificity. In this case, performance degradation was as great as 5.2% in recall and 4.7% in NPV.

These findings reinforce the conclusion that tailoring the classifier ratio is an effective method to adjust confidence in certain areas but comes at the expense of increasing risk in others. The effects are not equally experienced and may potentially expose a correlation between the distance of the web server from the reverse proxy and confidence levels. This introduces an intriguing characteristic to explore further.

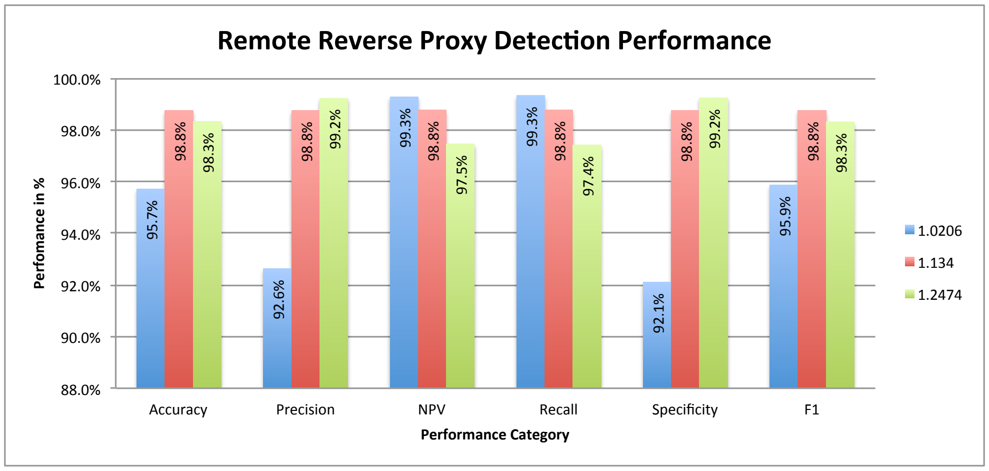


Figure 4.6: Graph of adjusted classifier performance on reverse proxies remotely connected to the web servers

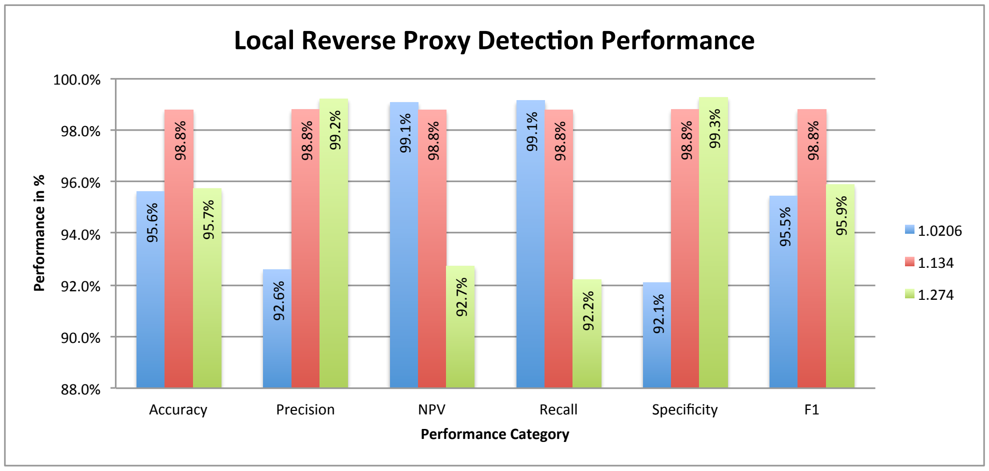


Figure 4.7: Graph of adjusted classifier performance on reverse proxies locally connected to the web servers

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

In order to fully trust the source of information that we receive, we must be confident that it is coming from whomever, or wherever, we expect. Current web requests are answered with data that may or may not be coming from the expected or intended source. Most times this is by design in efforts to improve performance or protect valuable web servers. Unfortunately, the same design aspects appeal to malicious actors by providing a method to launch attacks from behind the cover of intermediate devices. Because of this, we are vulnerable to all sorts of malicious activities and must be able to infer the topology in order to accurately calculate the inherent risk of our transactions.

5.1 Conclusions

A recent study found that there was a measurable difference in TCP RTT ratios from a client perspective when requesting web pages from servers located either behind a reverse proxy or accessed directly [3]. Our research discovered that not only is it measurable but that a precise classifier ratio can detect the presence of a reverse proxy server with nearly 99% accuracy.

To accomplish our goal we established a ground truth set of globally distributed PlanetLab nodes. This enabled us to know precisely the location and function of every client, reverse proxy, and web server leaving the only variability to the transport network of the Internet. We then gathered over 6 million TCP flows from which a machine learning technique was able to generate a specific, and optimal classifier ratio. Once the classifier was calculated we tested it against a unique subset of the TCP flows to determine its overall accuracy by using statistical cross-validation techniques.

We then made adjustments to the optimal classifier to determine the impact it would have on the ability to accurately or completely detect reverse proxies from a set of collected data. While performance suffered either way the classifier was adjusted, the most significant change was in precision and specificity when the classifier was decreased by 10%. Following that evaluation, we tested the adjusted classifiers against local, remote, and non-proxy connections. We found that the performance was the same despite the configuration when the classifier was decreased. Conversely, a large performance degradation was observed in when detecting local reverse proxy configurations when the classifier was increased.

While there were many lessons learned in conducting this research, the most significant was that timing analysis alone from only a client's perspective was sufficient to find an accurate classifier ratio and detect a reverse proxy. This contributes to many aspects of network measurements such as network mapping, security, and risk assessments for on-line transactions. With the knowledge of an accurate classifier we can more easily know whether our systems are receiving content from an original source or some other, possibly malicious or spoofed, device.

5.2 Future Work

- **Different Data Sets**
  Our method was applied to a single set of captured data using only PlanetLab nodes. While PlanetLab is a robust overlay network that is connected via the Internet, it is mostly homogenous in device types and may have contributed to the highly accurate classifier. Future work could apply our method to another data set captured from known clients, proxies, and web servers to see if it results in the same classifier ratio and accuracy.
- **Determine Actual Distance**
  An assumption can be made that legitimate users of reverse proxy servers locate them in approximately the same geographical location as the associated web server or servers. Likewise, one can assume that a reverse proxy that is a great distance from the origin web server is attempting to avoid detection. This research demonstrated that it is possible to detect the presence of a reverse proxy in the Internet. Additional research could potentially use this classifier to determine the actual distance or location of a reverse proxy server. Differences in timing ratios among different proxies and web servers may infer a difference in physical distance between devices.
- **Examine Time of Day Effects on Reverse Proxy Detection**
  Precautions were taken to avoid allowing time-of-day effects to skew our data collection. Tests were run in parallel where every node connected to every other node so that traffic congestion would not be isolated to only a portion of a single iteration. We

did not examine how time of day impacted measurements but did capture timing data in the form of pcap files and corresponding traceroute information. This extensive data set will allow for examination of time-of-day phenomena.

- **Use of Different HTTP messages**
  The only HTTP message used in this research was the GET message. Future work could identify whether other messages such as PUT, DELETE, or HEAD will result in the same timing differences.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] R. Soderbery. (2014, August). How many things are currently connected to the "Internet of things" (IOT)? [Online]. Available: http://www.forbes.com/sites/quora/2013/01/07/how-many-things-are-currently-connected-to-the-internet-of-things-iot/

[2] S. Corporation, "Internet security threat report - 2014," Symantec Corporation, Mountain View, CA, Tech. Rep., April 2014.

[3] M. Weant, "Fingerprinting reverse proxies using timing analysis of TCP flows," M.S. thesis, GSOIS, Naval Postgraduate School, Monterey, CA, 2013.

[4] M. Inc., *Merriam-Webster's collegiate dictionary*. Springfield, MA: Merriam-Webster, 2004.

[5] P. Institute, "2013 cost of cyber crime study: United States," Ponemon Institute, Traverse City, MI, Tech. Rep., October 2013.

[6] X. Hu, M. Knysz, and K. G. Shin, "Measurement and analysis of global IP-usage patterns of fast-flux botnets," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 2633–2641.

[7] J. Postel, "Transmission control protocol," Internet Engineering Task Force, Fremont, CA, RFC 793, September 1981.

[8] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 5th ed. New York: Addison-Wesley, 2010.

[9] R. Fielding and J. Reschke, "Hypertext transfer protocol (HTTP/1.1): Authentication," Internet Engineering Task Force, Fremont, CA, RFC 7230, June 2014.

[10] CNN. (2015, May). [Online]. Available: http://www.cnn.com

[11] Office for Civil Rights, HHS, "Standards for privacy of individually identifiable health information. Final rule." *Federal Register*, vol. 67, no. 157, p. 53181, 2002.

[12] J. Wack, K. Cutler, and J. Pole, "Guidelines on firewalls and firewall policy," National Institute of Standards and Technology, Gaithersburg, MD, Special Publication 800-41, September 2009.

[13] Alexa Internet, Inc. (2014, August). The top 500 sites on the web (global). [Online]. Available: http://www.alexa.com/topsites

[14] X. Hu, M. Knysz, and K. G. Shin, “RB-seeker: Auto-detection of redirection botnets,” in *NDSS Symposium*, 2009.

[15] A. R. Khakpour, J. W. Hulst, Z. Ge, A. X. Liu, D. Pei, and J. Wang, “Firewall fingerprinting,” in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 1728–1736.

[16] G. Lyon. (2014, August). Nmap: Free network security scanner. [Online]. Available: http://nmap.org

[17] T. Kohno, A. Broido, and K. C. Claffy, “Remote physical device fingerprinting,” *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, April-June 2005.

[18] D. Borman, R. Scheffenegger, and V. Jacobson, “TCP extensions for high performance,” Internet Engineering Task Force, Fremont, CA, RFC 7313, September 2014.

[19] Planetlab Consortium. (2014, August). An open platform for developing, deploying, and accessing planetary-scale services. [Online]. Available: http://www.planet-lab.org

[20] B. Chun, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, “PlanetLab: An overlay testbed for broad-coverage services,” *ACM SIGCOMM Computer Communications Review*, vol. 33, no. 3, pp. 3–12, July 2003.

[21] C. Dixon, T. Anderson, and A. Krishnamurthy, “Phalanx: Withstanding multimillion-node botnets,” in *NSDI ’08: 5th USENIX Symposium on Networked Systems Design and Implementation*, vol. 8. USENIX Association, 2008, pp. 45–58.

[22] K. B. Knight, J. Lawwill Jr., and B. L. Pulito, “Detecting a reverse proxy and establishing a tunneled connection therethrough,” September 2003, U.S. Patent 7,272,642.

[23] Google Maps. (2014, August). Planetlabs.

[24] vmware. (2015, May). [Online]. Available: http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=2055140

[25] I. Sysoev. (2014, November). [Online]. Available: http://www.nginx.org/en/

[26] D. Wessels. (2013, June). Squid caching proxy. [Online]. Available: http://www.squid-cache.org/

[27] The Apache Group. (2013, May). Apache - http server project. [Online]. Available: https://httpd.apache.org/

[28] L. MartinGarcia. (2015, February). TCPDUMP & LIBPCAP. [Online]. Available: http://www.tcpdump.org/

[29] H. Nikšić. (2015, February). GNU Wget. [Online]. Available: http://www.gnu.org/software/wget/

[30] A. Schulman and N. Spring, "Pingin' in the rain," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 19–28.

[31] S. Sundaresan, W. De Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè, "Broadband internet performance: a view from the gateway," *ACM SIGCOMM computer communication review*, vol. 41, no. 4, pp. 134–145, 2011.

[32] University of Ljublijana. (2015, February). Orange data mining - fruitful and fun. [Online]. Available: http://orange.biolab.si

[33] P. Bokoc. (2015, March). Fedora documentation. [Online]. Available: http://docs.fedoraproject.org/en-US/Fedora/20/html/Release_Notes/

[34] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Amsterdam: Centrum voor Wiskunde en Informatica, 1995.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California